CHƯƠNG 5
LẬP TRÌNH HỢP NGỮ LC-3

5.1 Lập trình hợp ngữ

5.2 Các thành phần của một chương trình hợp ngữ

5.3 Quá trình hợp dịch

5.4 Chương trình với nhiều modul

CHƯƠNG 5
LẬP TRÌNH HỢP NGỮ LC-3

5.1 Lập trình hợp ngữ

Hợp ngữ là một ngôn ngữ cấp thấp và là một bước nâng cấp nhỏ cho ISA của một máy tính. Mỗi lệnh hợp ngữ thường xác định một lệnh đơn trong ISA. Không như ngôn ngữ cấp cao, ngôn ngữ cấp thấp phụ thuộc rất nhiều vào ISA. Thực tế, ta sẽ thấy là mỗi kiến trúc tập lệnh ISA chỉ có duy nhất một hợp ngữ.

CHƯƠNG 5
LẬP TRÌNH HỢP NGỮ LC-3

5.2 Các thành phần của một chương trình hợp ngữ

Để hiểu rõ hợp ngữ LC-3, ta hãy xét chương trình ví dụ sau.

Ví dụ 5.1 : Chương trình nhân số nguyên với hằng số 6

```
01      ;
02      ;  Chương trình nhân một số với 6
03      ;
04                      .ORIG     x3050
05                      LD        R1, SIX
06                      LD        R2, NUMBER
07                      AND       R3, R3, #0      ; Xóa R3 để giữ tích
08                                                ; qua việc tính tổng cộng dồn
09      ;  Vòng lặp
0A      ;
0B      AGAIN           ADD       R3, R3, R2
0C                      ADD       R1, R1, #-1     ; biến theo dõi số lần lặp
```

CHƯƠNG 5
LẬP TRÌNH HỢP NGỮ LC-3

5.2 Các thành phần của một chương trình hợp ngữ

Để hiểu rõ hợp ngữ LC-3, ta hãy xét chương trình ví dụ sau.

```
0D                  BRp     AGAIN       ; lặp lại
0E      ;
0F                  HALT
10      ;
11      NUMBER      .BLKW   1
12      SIX         .FILL x0006
13      ;
14                  .END
```


Chương trình này nhân số nguyên được khởi tạo trong biến NUMBER với 6 bằng việc cộng số nguyên đó 6 lần. Ví dụ, nếu số nguyên đó là 123, chương trình sẽ tính tích bằng việc cộng 123 + 123 +123 + 123 + 123 + 123.

CHƯƠNG 5
LẬP TRÌNH HỢP NGỮ LC-3

5.2 Các thành phần của một chương trình hợp ngữ

5.2.1 Lệnh

Thay vì dùng dãy 16 bit 0 và 1 để biểu diễn một lệnh như trong trường hợp ISA LC-3, một lệnh hợp ngữ bao gồm bốn phần theo cấu trúc sau:

LABEL OPCODE OPERANDS *; COMMENTS*

Hai phần LABEL và COMMENTS là tùy chọn. Còn OPCODE và OPERANDS là bắt buộc.

CHƯƠNG 5
LẬP TRÌNH HỢP NGỮ LC-3

5.2 Các thành phần của một chương trình hợp ngữ

5.2.1 Lệnh:

1. Opcodes và Operands

Hai phần này phải có trong lệnh. Một lệnh phải quy định một mã thao tác OPCODE, tức là cái mà lệnh cần phải làm, và giá trị thích hợp của toán hạng OPERANDS, tức là cái mà lệnh sẽ dùng với tác vụ đã có. Đây là những thứ mà chúng ta đã gặp khi học LC-3.

OPCODE là tên tượng trưng cho mã tác vụ của lệnh LC-3 tương ứng. Với tên tượng trưng này, lập trình viên dễ dàng nhớ thao tác qua các tên như ADD, AND, hay LDR hơn là 4 bit 0001, 0101, hay 0110. Hình 4.3 liệt kê toàn bộ các OPCODES của 15 lệnh LC-3.

5.2 Các thành phần của một chương trình hợp ngữ

5.2.1 Lệnh

Số lượng các toán hạng phụ thuộc vào thao tác được thực thi. Ví dụ, lệnh ADD ở dòng 0B trong chương trình trên

```
AGAIN   ADD   R3, R3, R2
```

Lệnh LD ở dòng 06

```
        LD    R2, NUMBER
```

Trong trường hợp toán hạng tức thời, các giá trị thực cần được ghi rõ trong lệnh (như trị 0 trong dòng 07).

```
        AND   R3, R3, #0      ; xóa R3 để giữ tích
```

Chúng ta dùng dấu # cho số thập phân, x cho thập lục phân, và b cho nhị phân.

CHƯƠNG 5
LẬP TRÌNH HỢP NGỮ LC-3

5.2 Các thành phần của một chương trình hợp ngữ

5.2.1 Lệnh

2. Nhãn

Nhãn là các tên tượng trưng được dùng để xác định các ô nhớ được tham khảo tới trong chương trình. Trong hợp ngữ LC-3, một nhãn có thể được tạo từ một tới 20 ký số hay ký tự, và bắt đầu bằng một ký tự, như LAPLAI, KETTHUC, LAP100,….

Có hai lý do cần cho việc tham khảo một vị trí trong bộ nhớ, đó là

- Ô nhớ vị trí đó chứa đích của một lệnh rẻ nhánh, ví dụ AGAIN trong dòng 0B.

- Ô nhớ vị trí đó chứa một giá trị cần được nạp hay lưu, ví dụ, NUMBER ở dòng 11, và SIX ở dòng 12.

5.2 Các thành phần của một chương trình hợp ngữ

5.2.1 Lệnh

Vị trí tương ứng nhãn AGAIN được tham khảo bởi lệnh rẽ nhánh ở dòng 0E,

BRp AGAIN

5.2 Các thành phần của một chương trình hợp ngữ

5.2.1 Lệnh

3. Ghi chú

Ghi chú là các thông điệp chỉ cần thiết với con người. Các ghi chú không có bất kỳ ảnh hưởng nào trong quá trình dịch và cũng không chịu tác động nào từ bộ dịch hợp ngữ LC-3. Chúng được quy định trong chương trình bằng các dấu chấm phẩy đặt trước, phần sau dấu chấm phẩy (nếu có) là một ghi chú và được bộ dịch bỏ qua.

CHƯƠNG 5
LẬP TRÌNH HỢP NGỮ LC-3

5.2 Các thành phần của một chương trình hợp ngữ

5.2.2 Mã giả (Các hướng dẫn dịch)

Bộ hợp dịch LC-3 là một chương trình lấy đầu vào là chuỗi ký tự biểu diễn một chương trình được viết bằng hợp ngữ LC-3, và dịch nó ra thành một chương trình ở cấp kiến trúc tập lệnh (ISA) của LC-3.

Mã giả (pseudo-ops) giúp cho bộ dịch thực hiện nhiệm vụ này, còn được gọi bằng một tên khác là hướng dẫn dịch (assembler directives).

Bộ hợp dịch LC-3 gồm năm mã giả: .ORIG, .FILL, .BLKW, .STRINGZ, và .END. Tất cả mã giả này đều có dấu chấm như là ký tự đầu tiên của nó.

5.2 Các thành phần của một chương trình hợp ngữ

5.2.2 Mã giả (Các hướng dẫn dịch)

.ORIG

.ORIG cho bộ dịch biết nơi bắt đầu chương trình LC-3 trong bộ nhớ. Ở dòng 04, .ORIG x3050 nói rằng, chương trình bắt đầu ở vị trí x3050. Và tất nhiên, lệnh LD R1, SIX sẽ được đặt ở vị trí x3050.

.FILL

.FILL nói cho bộ hợp dịch biết việc cần dùng vị trí kế trong chương trình (và tất nhiên là sau này là bộ nhớ khi chạy chương trình), và khởi động nó bằng giá trị của toán hạng. Ở dòng 12, vị trí thứ 9 (tính từ lệnh đầu tiên) trong chương trình LC-3 được khởi động trị x0006.

5.2 Các thành phần của một chương trình hợp ngữ

5.2.2 Mã giả (Các hướng dẫn dịch)

.BLKW

.BLKW bắt bộ dịch để dành một số ô nhớ (tức BLocK Words) trong chương trình. Số ô nhớ thực sự là toán hạng của mã giả .BLKW. Ở dòng 11, mã giả yêu cầu bộ dịch để dành một ô nhớ với nhản là NUMBER.

.STRING

.STRING bắt bộ dịch khởi tạo một chuỗi n +1 ô nhớ. Đối số là dãy n ký tự, bên trong cặp dấu nháy kép. Khi đó, n từ nhớ đầu tiên được khởi động bằng các ký tự mã ASCII 8 bit được mở rộng zero (để có 16 bit) trong chuỗi. Từ nhớ cuối cùng được khởi tạo là 0, tức x0000, là trị canh để truy xuất chuỗi các mã ASCII.

5.2 Các thành phần của một chương trình hợp ngữ

5.2.2 Mã giả (Các hướng dẫn dịch)

Ví dụ 5.2: Đoạn mã sau:

```
        .ORIG       x3010
HELLO   .STRINGZ    “Hello, World!”
```

5.2 Các thành phần của một chương trình hợp ngữ

5.2.2 Mã giả (Các hướng dẫn dịch)

x3010: x0048
x3011: x0065
x3012: x006C
x3013: x006C
x3014: x006F
x3015: x002C
x3016: x0020
x3017: x0057
x3018: x006F
x3019: x0072
x301A: x006C
x301B: x0064
x301C: x0021
x301D: x0000

5.2 Các thành phần của một chương trình hợp ngữ

5.2.2 Mã giả (Các hướng dẫn dịch)

.END

.END nói cho bộ dịch biết chương trình kết thúc ở đâu. Bất kỳ ký tự nào đứng sau .END sẽ bị bộ hợp dịch bỏ qua. Như vậy, thực ra .END chỉ đơn giản là một quy định giới hạn, nó đánh dấu sự kết thúc của chương trình nguồn.

5.2 Các thành phần của một chương trình hợp ngữ

5.2.3 Một ví dụ

Trong mục này chúng ta xét lại ví dụ ở mục 4.10, tính số lần xuất hiện của một ký tự trong một file cho trước. Ký tự cần kiểm tra được vào từ bàn phím, file ký tự được xem là mảng ký tự cần được khởi tạo trước khi chạy chương trình. Giải thuật ở dạng lưu đồ và chương trình ở dạng ISA LC-3 được trình bày trong hình 4.18 và 4.19.

5.2 Các thành phần của một chương trình hợp ngữ

5.2.3 Một ví dụ

Ví dụ 5.3: Chương trình đếm số lần xuất hiện ký tự trong file ở dạng hợp ngữ LC-3.

```
01    ;
02    ; Chương trình đếm số lần xuất hiện ký tự trong file.
03    ; Ký tự cần kiểm tra được nhập từ bàn phím.
04    ; Kết quả được hiển thị ra màn hình.
05    ;Chương trình chỉ làm việc đúng khi số lần xuất hiện ký tự không quá 9.
06    ;
07    ;
08    ; Khởi động
09    ;
0A              .ORIG x3000
0B              AND  R2, R2, #0      ; R2 là bộ đếm, được khởi động bằng 0
0C              LD   R3, PTR         ; R3 là pointer tới các ký tự trong file
0D              TRAP x23                    ; R0 lưu ký tự được nhập vào
0E              LDR  R1, R3, #0      ; R1 giữ ký tự đầu tiên
```

5.2 Các thành phần của một chương trình hợp ngữ

5.2.3 Một ví dụ

```
0F   ;
10   ; Kiểm tra ký tự kết thúc file EOT
11   ;
12   TEST     ADD   R4, R1, #-4    ; Kiểm tra EOT (ASCII x04)
13            BRz   OUTPUT         ; Nếu đúng, chuẩn bị xuất
14   ;
15   ; Kiểm tra sự thích hợp của ký tự. Nếu có, tăng biến đếm.
16   ;
17            NOT   R1, R1
18            ADD   R1, R1, R0     ; Nếu đúng là ký tự cần kiểm tra, R1 = xFFFF
19            NOT   R1, R1         ; Nếu đúng là ký tự cần kiểm tra, R1 = x0000
1A            BRnp  GETCHAR        ; Nếu không đúng, không tăng đếm, lấy ký tự kế
1B            ADD   R2, R2, #1
1C   ;
1D   ; Lấy ký tự kế tiếp trong file.
1E   ;
1F   GETCHAR  ADD       R3, R3, #1      ; Chỉ tới ký tự kế.
20            LDR   R1, R3, #0     ; R1 lấy ký tự kế để kiểm tra
21            BRnzp TEST
22   ;
```

5.2 Các thành phần của một chương trình hợp ngữ

5.2.3 Một ví dụ

```
23    ; Xuất số lần xuất hiện.
24    ;
25    OUTPUT   LD  R0, ASCII     ; Nạp mẫu ASCII
26             ADD  R0, R0, R2   ; Chuyển số trị từ nhị phân ra ký tự ASCII
27             TRAP x21          ; Mã ASCII trong R0 được hiển thị.
28             TRAP x25          ; Chấm dứt chương trình.
29    ;
2A    ; Phần lưu trữ pointer và mẫu ASCII
2B    ;
2C    ASCII    .FILL  x0030
2D    PTR      .FILL  x4000
2E             .END
```

CHƯƠNG 5
LẬP TRÌNH HỢP NGỮ LC-3

5.3 Quá trình hợp dịch

5.3.1 Giới thiệu

Trước khi một chương trình hợp ngữ LC-3 được thực thi, nó phải được dịch ra thành một chương trình ngôn ngữ máy, có nghĩa là từng lệnh trong đó sẽ là từng lệnh ở ISA LC-3. Đây là công việc của bộ dịch hợp ngữ LC-3.

Với bộ hợp dịch LC-3 (mà chúng ta có thể download từ mạng), ta có thể dịch từ chương trình hợp ngữ ra chương trình ngôn ngữ máy. Trong giáo trình này, các chương trình hợp ngữ có thể được viết và được dịch ra dạng ISA bằng LC-3 Editor mà chúng ta có thể tìm thấy trên mạng.

5.3 Quá trình hợp dịch

5.3.2 Quá trình dịch

Gồm 2 giai đoạn:

- Tạo bảng biểu, Constructing table
- Dịch ra ngôn ngữ máy (nhị phân)

CHƯƠNG 5
LẬP TRÌNH HỢP NGỮ LC-3

5.3 Quá trình hợp dịch

5.3.3 Bước đầu tiên: Tạo bảng biểu trưng

Bảng biểu trưng là một sự tương ứng giữa các tên tượng trưng với các địa chỉ 16 bit của chúng tính từ đầu chương trình. Nên nhớ rằng, chúng ta cần các nhãn ở nhưng chổ cần được tham khảo, hoặc đó là đích của một lệnh rẻ nhánh hoặc nơi đó chứa dữ liệu cần được nạp hay lưu. Vì vậy, nếu chúng ta không có bất kỳ một lỗi lập trình nào, và nếu chúng ta xác định được tất cả các nhãn, chúng ta hẳn sẽ xác định được tất cả các địa chỉ tượng trưng được dùng trong chương trình.

5.3 Quá trình hợp dịch

5.3.3 Bước đầu tiên: Tạo bảng biểu trưng

Lệnh đầu tiên có nhãn là lệnh ở dòng 12. Vì nó là lệnh thứ năm của chương trình, nên lúc này LC chứa x3004, một đầu vào trong bảng biểu trưng được tạo ra như sau:

Symbol	Address
TEST	x3004

Lệnh thứ hai có nhãn là lệnh ở dòng 1F. Tại đây, LC đã được tăng lên tới x300B. Như vậy một đầu vào trong bảng đã được tạo ra thêm như sau:

Symbol	Address
GETCHAR	x300B

5.3 Quá trình hợp dịch

5.3.3 Bước đầu tiên: Tạo bảng biểu trưng

Tới lúc cuối của bước dịch đầu tiên, bảng biểu trưng có các đầu vào như sau:

Symbol	Address
TEST	x3004
GETCHAR	x300B
OUTPUT	x300E
ASCII	x3012
PTR	x3013

5.3 Quá trình hợp dịch

5.3.4 Bước thứ hai: Dịch ra ngôn ngữ máy

Bước dịch thứ hai gồm việc duyệt qua chương trình hợp ngữ lần thứ hai, theo từng dòng, lúc này với sự trợ giúp của bảng biểu trưng. Ở mỗi dòng, lệnh hợp ngữ được dịch ra lệnh ngôn ngữ máy LC-3.

5.3 Quá trình hợp dịch

5.3.4 Bước thứ hai: tạo chương trình ngôn ngữ máy

Lần này, khi bộ dịch lấy lệnh ở dòng 0C, nó có thể hoàn toàn dịch lệnh này vì nó biết nhãn PTR tương ứng với x3013. Lệnh là LD, có opcode là 0010. Thanh ghi đích là R3, nghĩa là 011.

PCoffset được tính như sau: chúng ta biết rằng PTR là nhãn cho địa chỉ x3013, và thanh ghi PC đã tăng là LC + 1, tức x3002. Vì PTR (x3013) phải là tổng của PC đã tăng (x3002) và PCoffset được mở rộng dấu, nên PCoffset phải là x0011. Ghép tất cả điều này lại với nhau, ta thấy lệnh ở x3001 là 0010011000010001, và LC được tăng lên x3002.

5.3 Quá trình hợp dịch

5.3.4 Bước thứ hai: Dịch ra ngôn ngữ máy

Chương trình đã được dịch và nhận được là

Address	Binary
	0011000000000000
x3000	0101010010100000
x3001	0010011000010001
x3002	1111000000100011
x3003	0110001011000000
x3004	0001100001111100
x3005	0000010000001000
x3006	1001001001111111
x3007	0001001001000000
x3008	1001001001111111
x3009	0000101000000001
x300A	0001010010100001
x300B	0001011011100001
x300C	0110001011000000
x300D	0000111111110110
x300E	0010000000000011
x300F	0001000000000010
x3010	1111000000100001
x3011	1111000000100101
x3012	0000000000110000
x3013	0100000000000000

CHƯƠNG 5
LẬP TRÌNH HỢP NGỮ LC-3

5.4 Chương trình với nhiều modul

5.4.1 Bản thực thi

Khi máy tính bắt đầu thực thi một chương trình, tập tin thực thi của chương trình được gọi là bản thực thi (Executable image). Bản thực thi thường được tạo ra từ nhiều modul do nhiều lập trình viên thiết kế ra một cách độc lập. Mỗi modul được dịch một cách riêng biệt và tạo thành một tập tin đối tượng (object). Nếu các modul được viết bằng hợp ngữ LC-3, chúng sẽ được dịch bằng bộ dịch hợp ngữ LC-3. Những modul được viết bằng C sẽ được dịch bằng bộ dịch C. Có những modul do lập trình viên viết khi thiết kế chương trình, và cũng có những modul là các chương trình con được cung cấp bởi hệ điều hành. Mỗi tập tin đối tượng bao gồm các lệnh trong kiến trúc tập lệnh (ISA) của máy tính đang được sử dụng, cùng với các dữ liệu liên quan.

5.4 Chương trình với nhiều modul

5.4.1 Bản thực thi

Bước cuối cùng là liên kết (link) tất cả các modul lại với nhau để có một tập tin gọi là bản thực thi. Trong suốt quá trình thực thi, các chu kỳ lệnh FETCH, DECODE, … được áp dụng cho các lệnh trong bản thực thi.

5.4 Chương trình với nhiều modul

5.4.2 Thiết kế với nhiều tập tin đối tượng

Khi thiết kế một chương trình, chúng ta thường dùng thư viện của hệ điều hành cũng như các modul được viết bởi các lập trình viên khác trong nhóm. Do đó, việc bản thực thi được tạo ra từ nhiều tập tin đối tượng khác nhau là rất phổ biến.

Trong chương trình ví dụ đếm số ký tự xuất hiện trong một tập tin là mảng, ta có thể thấy một áp dụng tiêu biểu của chương trình với hai modul, gồm modul chương trình và modul là tập tin dữ liệu. Với ví dụ 5.3, địa chỉ bắt đầu của tập tin mảng dữ liệu là x4000 ở dòng 2D không được quan tâm khi chương trình được viết.

5.4 Chương trình với nhiều modul

5.4.2 Thiết kế với nhiều tập tin đối tượng

Nếu chúng ta thay thế dòng 2D này bằng

```
PTR     .FILL     STARTofFILE
```

thì chương trình ví dụ này sẽ không được hợp dịch vì không có đầu vào cho STARTofFILE trong bảng biểu trưng. Chúng ta giải quyết việc này ra sao ?

Mặt khác, nếu hợp ngữ LC-3 có mã giả .EXTERNAL, chúng ta có thể xác định STARofFILE như là một tên biểu trưng của một địa chỉ không được biết lúc chương trình 5.3 được dịch.

Điều này có thể được thực hiện bằng dòng sau

```
.EXTERNAL    STARTofFILE
```

5.4 Chương trình với nhiều modul

5.4.2 Thiết kế với nhiều tập tin đối tượng

Mà sẽ báo cho bộ dịch LC-3 rằng sự vắng mặt của nhãn STARTofFILE không phải là một lỗi trong chương trình. Hơn nữa, STARTofFILE là một nhãn trong modul khác và modul này sẽ được dịch một cách độc lập. Trong ví dụ 5.3, đó chính là nhãn của vị trí của ký tự đầu tiên trong tập tin mảng mà sẽ được chương trình đếm ký tự của chúng ta khảo sát.

5.4 Chương trình với nhiều modul

5.4.2 Thiết kế với nhiều tập tin đối tượng

Nếu hợp ngữ LC-3 có được mã giả .EXTERNAL, và nếu chúng ta đã thiết kế nhãn STARTofFILE theo .EXTERNAL, LC-3 có khả năng tạo một đầu vào trong bảng biểu trưng cho STARTofFILE, và thay vì gán cho nhãn này một địa chỉ, LC-3 sẽ đánh dấu biểu trưng tùy thuộc modul khác. Lúc liên kết, khi tất cả các modul được kết nối lại, bộ liên kết (tức chương trình phụ trách việc nối này) sẽ dùng đầu vào cho STARTofFILE trong bảng biểu trưng trong modul khác để hoàn tất việc dịch dòng 2D.

5.4 Chương trình với nhiều modul

5.4.2 Thiết kế với nhiều tập tin đối tượng

Theo cách này, mã giả .EXTERNAL cho phép việc tham khảo của một modul tới các vị trí biểu trưng trong một modul khác một cách dễ dàng. Quá trình dịch phù hợp được bộ liên kết giải quyết.